**基于QT的多线程Socket编程**

目录

[一．前言 2](#_Toc121752651)

[二．总体设计 3](#_Toc121752652)

[1 整体框架 3](#_Toc121752653)

[2 功能设计 3](#_Toc121752654)

[3 流程设计 6](#_Toc121752655)

[3.1服务端 6](#_Toc121752656)

[3.2客服端 7](#_Toc121752657)

[三．详细设计 8](#_Toc121752658)

[1. 服务端 8](#_Toc121752659)

[1.1类设计 8](#_Toc121752660)

[1.2详细设计 9](#_Toc121752661)

[2. 客服端 10](#_Toc121752662)

[2.1类设计 10](#_Toc121752663)

[2.2详细设计 11](#_Toc121752664)

[四．测试 13](#_Toc121752665)

[五．总结 15](#_Toc121752666)

[六．附录 16](#_Toc121752667)

[1.server源代码 16](#_Toc121752668)

[2.Client源代码 24](#_Toc121752669)

# 一．前言

Qt 是一个1991年由Qt Company开发的跨平台C++图形用户界面应用程序开发框架。它既可以开发GUI程序，也可用于开发非GUI程序，比如控制台工具和服务器。Qt是面向对象的框架，使用特殊的代码生成扩展（称为元对象编译器(Meta Object Compiler, moc)）以及一些宏，Qt很容易扩展，并且允许真正地组件编程。

Qt network模块中为我们提供了QTcpServer Class以及QTcpSocket Class。通过这两个模块，可以实现进程之间的通信。我们可以把Server和Client部署到两台主机中，Client主机中创建Socket绑定Server服务所在主机的ip和端口号，Server主机监听对应的端口号，实现进程之间的通信。

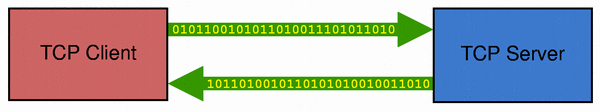


图 1 套接字通信

# 总体设计

## 整体框架

程序的整体框架采用Qt的network框架完成。客户端通过绑定ip和端口号，服务端监听端口号。当服务端监听成功后，会产生一个Socket,这个套接字供给服务端实用，以完成对数据的读、写操作。而客户端，默认自带套接字，不需要在生成。



图 2总体框架

## 功能设计

本程序的功能是在同一局域网下，实现不同或相同主机的进行之间的通信。在同一局域网下，客户端向服务端传递信息的过程如下：



图 3局域网

了解了局域网下，客户端与服务端的通常流程后，下面给出了具体的功能设计。其中包括单向通信、多线程管理、文件传输、可靠通信。



图 4功能设计

## 流程设计

### 3.1服务端



图 5服务端流程

### 3.2客服端



图 6客户端流程

# 详细设计

## 服务端

### 1.1类设计



图 7 QTcpServer类



图 8 Work类

### 1.2详细设计

#### 1.2.1 Server类

Server类的设计比较简单，它主要是用于绑定并监听端口的任务。其中，最重要的就是重载了其父类的函数incomingConnection。通过这个函数，可以得到一个已接受连接的本机套接字描述符。

多线程的应用场景如图 9所示。当服务端收到多个客户端的连接请求时，触发incomingConnection函数，在函数中就会通过QThread\* **t** = new QThread;方法，产生多个子线程来完成不同客户端的工作。



图 9 多线程示例

表 1多线程代码示例

|  |
| --- |
| void Server::***incomingConnection***(qintptr **handle**)  {  qInfo() <<"有新的消息来了：";  //创建子进程  QThread\* **t** = new QThread;  Work\* **worker** = new Work(handle);  worker->moveToThread(*t*);  t->start();  } |

#### Work类

Work类是用于处理客服端发来的数据的类。在Server类中，通过worker->moveToThread(*t*)方法，把work类的控制权移交给子线程t所控制。

在work类中，通过自定的数据包的打包协议进行拆包，最后把完整的内容保存在服务端中存储。

## 客服端

### 2.1类设计



图 10 Widget类



图 11work类

### 2.2详细设计

#### 2.2.1 Widget类设计

Widget类的主要功能包括建立客服端UI界面、从UI界面中获得服务器、创建子线程实现文件传输。

其UI的设计如下图所示。

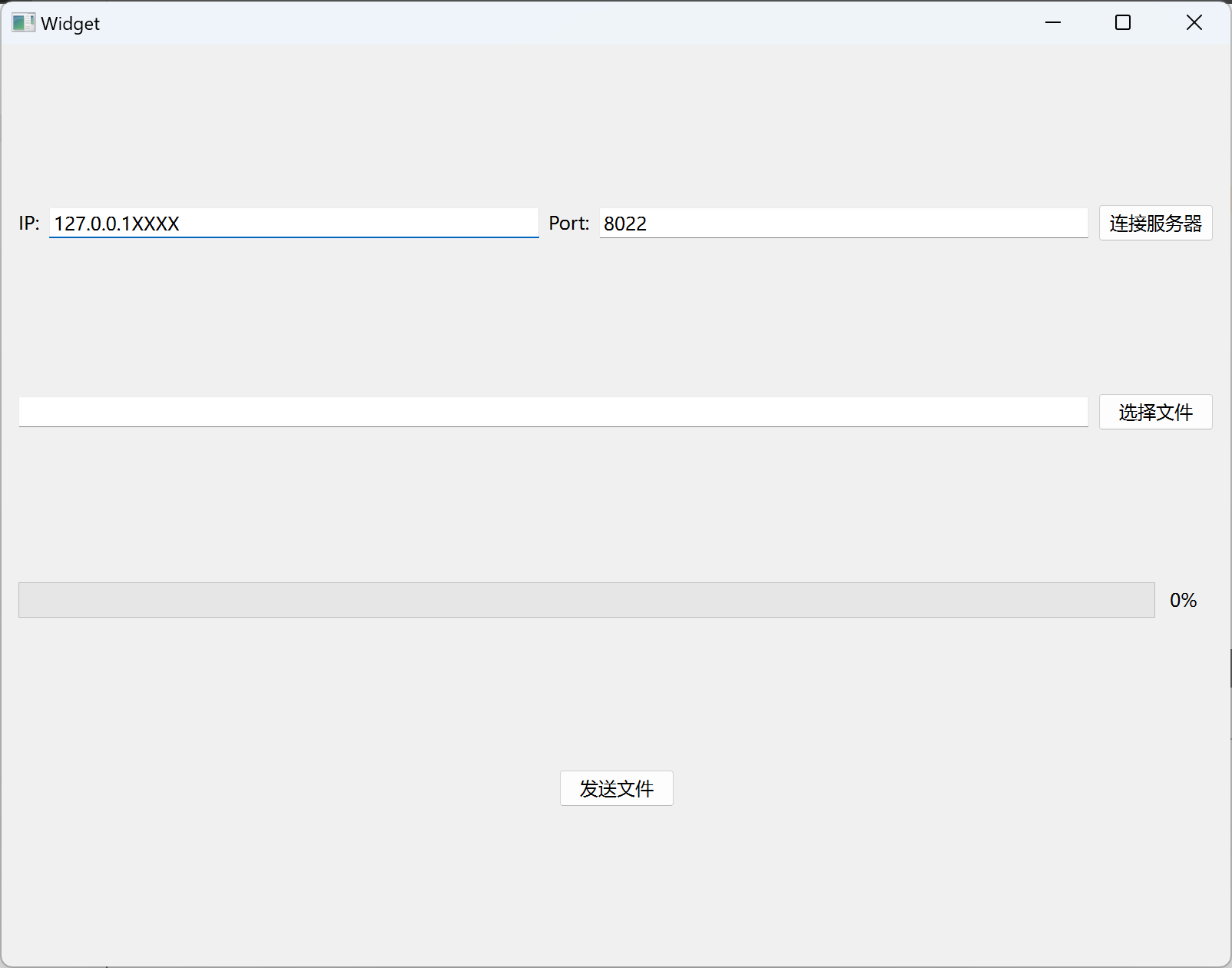


图 12 UI设计

#### work类

Work类，继承于QObject类，它是一个工作类，作为在Widget类中创建的子线程的一个工作类，通过worker->moveToThread(*t*)，来实现。

这里的主要难点就是在于数据的打包。约定一个数据包的格式如下：

表 2数据包格式

|  |
| --- |
| typedef struct **packet**  {  int **len**;  char **buffer**[4096];  }**NetWorkPacket**; |

既，我们约定一个数据包的格式为数据的字节长度和数据本身，当buffer装不下数据时，可以分段发送。

# 测试

通过两台主机来进行测试，其中Server主机的ip地址为192.168.31.129，Client主机的ip地址为192.168.31.90。

在Client主机中，开辟3个客户端进行多线程的Sotcket的测试。

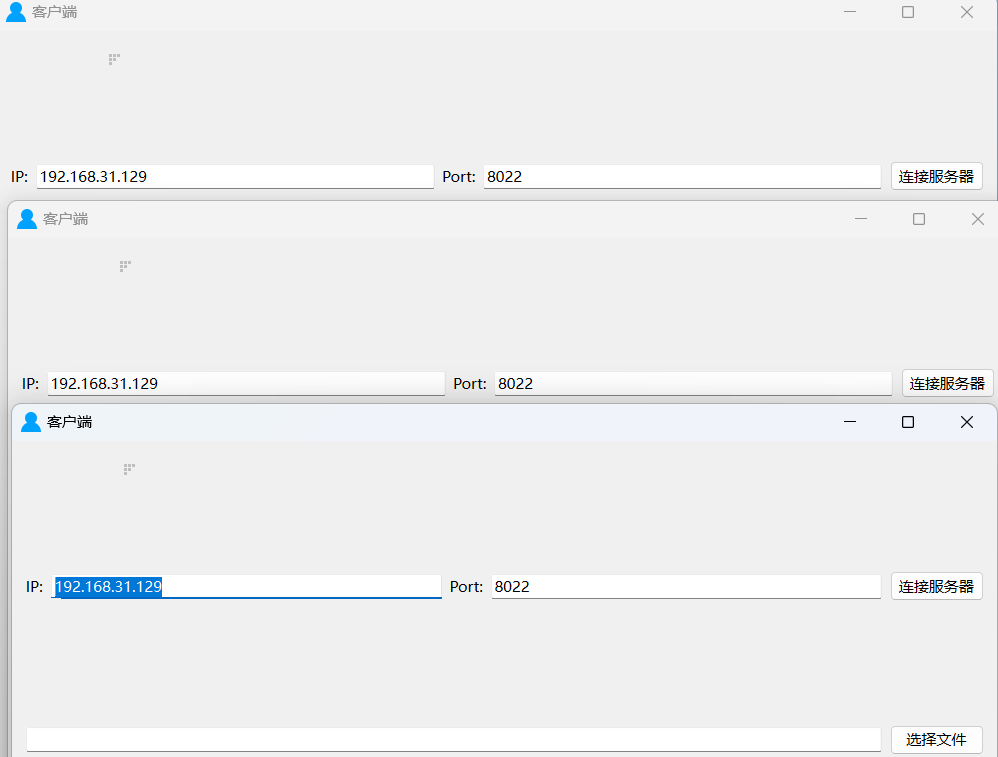
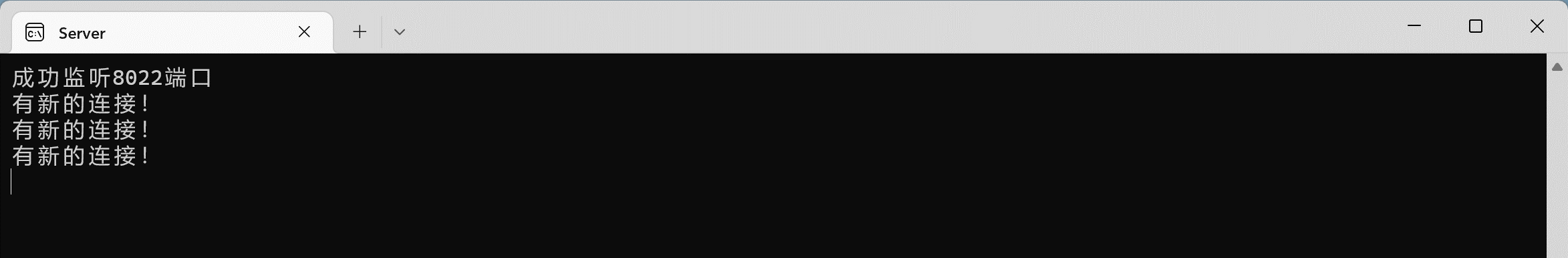


图 13 客户端

这时，在服务端可以看到有三个新的客户端连接成功！



接下来，进行数据传送的测试:

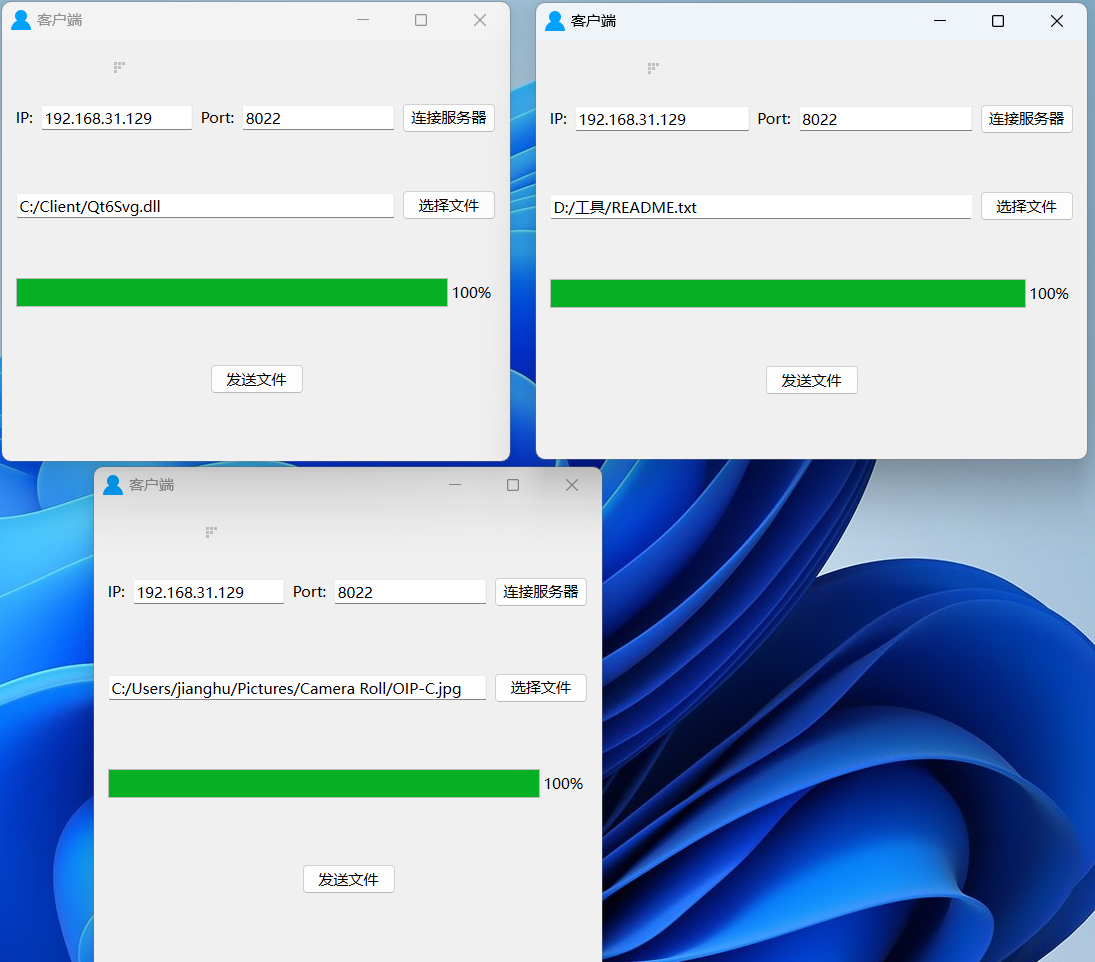


图 14 客户端发送数据

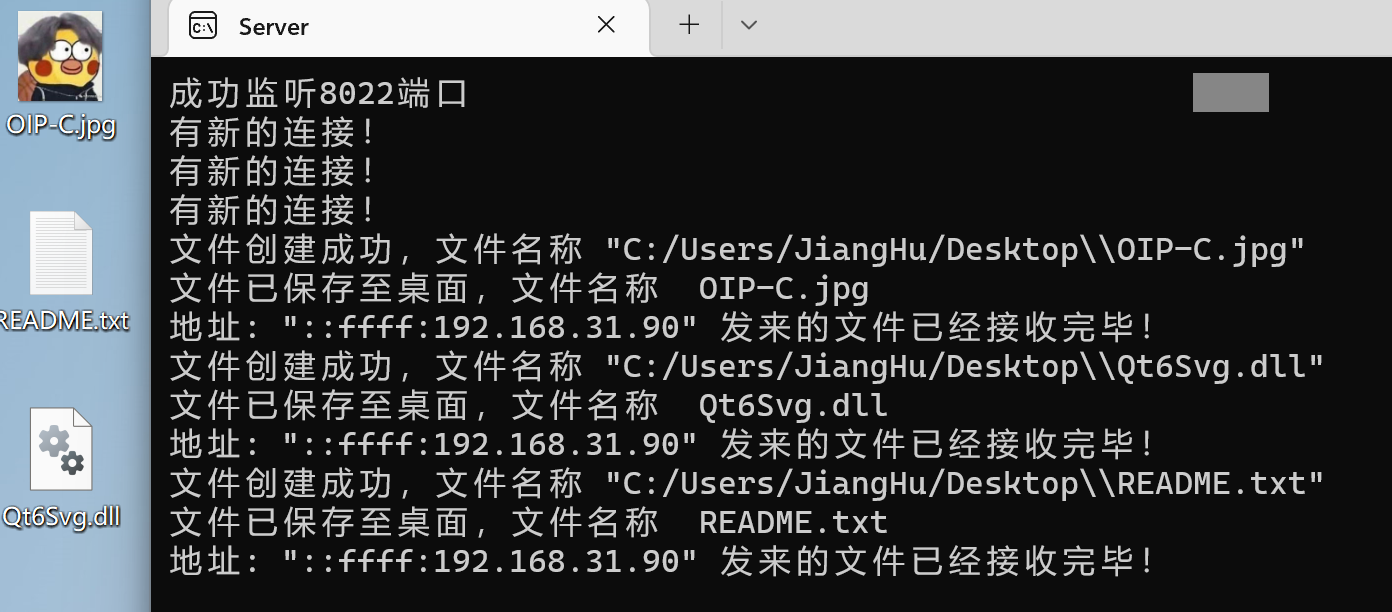


图 15服务接收数据

从中可以看到，Client主机打开了三个客户端，向Server中发送了三个不同的文件。在Server中，对应的就会生产三个子线程去接受Client发来的三个文件，并保存到桌面上。

# 总结

套接字Socket看做是不同主机之间的进程进行双向通信的端点，简单的说就是通信的两方的一种约定，用套接字中的相关函数来完成通信过程。套接字Socket是连接应用程序和网络驱动程序的桥梁，套接字Socket在应用程序中创建，通过绑定与网络驱动建立关系。此后，应用程序送给套接字Socket的数据，由套接字Socket交给网络驱动程序向网络上发送出去。计算机从网络上收到与该套接字Socket绑定IP地址和端口号相关的数据后，由网络驱动程序交给Socket，应用程序便可从该Socket中提取接收到的数据，网络应用程序就是这样通过Socket进行数据的发送与接收的。

**操作系统区分不同应用程序进程间的网络通信和连接,主要有3个参数：通信的目的IP地址、使用的传输层协议(TCP或UDP)和使用的端口号。**

在多线程的优势很明显，譬如在本软件的Client端的应用。Client端中既包括UI和数据处理的工作。想象这样的一个应用场景，在UI线程的代码中，加入一个很长的循环，那么这个UI界面就会卡住一段时间。如果把UI的代码和工作的代码分为两个线程，那么就会被产生UI卡的问题。

**多线程可以提高CPU的利用率。在多线程程序中，一个线程必须等待的时候，CPU可以运行其它的线程而不是等待，这样就大大提高了程序的效率。**

# 附录

## 1.server源代码

|  |
| --- |
| Server.h  #ifndef SERVER\_H  #define **SERVER\_H**  #include <QTcpServer>  #include <QTcpSocket>  #include <QThread>  #include <work.h>  class **Server** : public QTcpServer  {  public:  **Server**(QObject \***parent** = nullptr);  QTcpServer \***m\_server**;  protected:  void ***incomingConnection***(qintptr **handle**) override;  };  #endif // SERVER\_H |
| Server.cpp  #include "server.h"  Server::**Server**(QObject \***parent**) :  QTcpServer(*parent*)  {  }  void Server::*incomingConnection*(qintptr handle)  {  qInfo() <<"有新的消息来了：";  //创建子进程  QThread\* **t** = new QThread;  Work\* **worker** = new Work(handle);  worker->moveToThread(*t*);  t->start();  } |
| Work.h  #ifndef WORK\_H  #define **WORK\_H**  #include <QObject>  #include <QTcpSocket>  #include <QFile>  #include <QStandardPaths>  class **Work** : public QObject  {  Q\_OBJECT  public:  explicit **Work**(qintptr **socketDescriptor**,QObject \***parent** = nullptr);  QTcpSocket\* **m\_socket**;  QFile\* **file**;  int **nOffset**;  int **num**;  int **len**;  public:  void **save**(const char\* **content**);  void **openFile**(const char\* **type**);  signals:  public slots:  void **OnReadyRead**();  void **OnConnected**();  void OndisConnected();  };  #endif // WORK\_H |
| Work.cpp  #include "work.h"  Work::**Work**(qintptr **socketDescriptor**, QObject \***parent**)  : QObject{*parent*}  {  num = 1;  nOffset = 0;  m\_socket = new QTcpSocket;  m\_socket->*setSocketDescriptor*(socketDescriptor);  connect(m\_socket, &QTcpSocket::readyRead, this, &Work::OnReadyRead);  connect(m\_socket, &QTcpSocket::connected, this, &Work::OnConnected);  connect(m\_socket, &QTcpSocket::disconnected, this, &Work::OndisConnected);  }  void Work::**save**(const char \***content**)  {  file->*open*(QIODevice::ReadWrite);  if(file->isOpen())  {  file->write(content);  qInfo() << "写入完毕!";  }  else  {  qInfo() << "文件不能打开";  }  file->*close*();  delete file;  }  void Work::**openFile**(const char \***type**)  {  QString **path** = QStandardPaths::writableLocation(QStandardPaths::DesktopLocation) + "\\save." +type;  qInfo() << path;  file = new QFile(path);  file->*open*(QIODevice::ReadWrite);  file->*close*();  }  void Work::**OnReadyRead**()  {  int **n** = m\_socket->*bytesAvailable*(); //接收到的字节数  qInfo() << "大小：" << n;  char\* **buffer** = (char\*) malloc (sizeof(char) \* n);  m\_socket->read(*buffer*, n);  if(num == 1)  {  memcpy(&*len*, buffer, sizeof(int)); //包头：包体长度  nOffset += sizeof(int);  char\* **tmpBuf** = (char\*) malloc (sizeof(char) \* len + 1);  memcpy(tmpBuf + len,"\0",1);  memcpy(*tmpBuf*, buffer+nOffset, len); //包体  nOffset += len;  //类别  openFile(tmpBuf);  }  memcpy(&*len*, buffer+nOffset, sizeof(int));  nOffset += sizeof(int);  char\* **tmpBuf** = (char\*) malloc (sizeof(char) \* len + 1);  memcpy(tmpBuf + len,"\0",1);  memcpy(*tmpBuf*, buffer+nOffset, len);  nOffset += len;  qInfo() << tmpBuf;  save(tmpBuf);  free(*buffer*);  }  void Work::**OnConnected**()  {  qInfo() << "已连接客服端：" << m\_socket->peerName() << " 地址：" << m\_socket->peerAddress()  <<" 端口：" <<m\_socket->peerPort();  }  void Work::OndisConnected()  {  qInfo() << "客服端：" << m\_socket->peerName() << "以下线 地址：" << m\_socket->peerAddress()  <<" 端口：" <<m\_socket->peerPort();  m\_socket->*close*();  m\_socket->deleteLater();  } |

|  |
| --- |
| Main.cpp  #include <QCoreApplication>  #include <server.h>  int main(int argc, char \*argv[])  {  QCoreApplication a(argc, argv);  Server s;  if (s.listen(QHostAddress::Any, 8022))  {  qInfo() << "成功监听8022端口";  }  else  {  qInfo() << "监听失败";  }  return a.exec();  } |

## 2.Client源代码

|  |
| --- |
| Widget.h:  #ifndef WIDGET\_H  #define **WIDGET\_H**  #include <QWidget>  #include <work.h>  #include <QThread>  #include <QMessageBox>  #include <QFileDialog>  QT\_BEGIN\_NAMESPACE  namespace **Ui** { class **Widget**; }  QT\_END\_NAMESPACE  class **Widget** : public QWidget  {  Q\_OBJECT  public:  **Widget**(QWidget \***parent** = nullptr);  ~**Widget**();  void **OnFileOpenError**();  private slots:  void **on\_setFile\_2\_clicked**();  void **on\_setFile\_clicked**();  void **on\_send\_clicked**();  void **setValue**(int **value**);  signals:  void **startConnect**(QString,unsigned short);  void **sendSignal**(QString **path**);  private:  Ui::Widget \***ui**;  };  #endif // WIDGET\_H |
| Widget.cpp  #include "widget.h"  #include "ui\_widget.h"  Widget::Widget(QWidget \*parent)  : QWidget(parent)  , ui(new Ui::Widget)  {  ui->setupUi(this);  ui->ip->setText("127.0.0.1");  ui->Port->setText("8022");  ui->progressBar->setRange(0,100);  ui->progressBar->setValue(0);  //创建子进程  QThread\* t = new QThread;  Work\* worker = new Work;  worker->moveToThread(t);  connect(this,&Widget::startConnect,worker,&Work::connectServer);  connect(worker, &Work::curPercent, ui->progressBar, &QProgressBar::setValue);  connect(worker, &Work::canNotOpenFile, this, &Widget::OnFileOpenError);  connect(this,&Widget::sendSignal, worker, &Work::send);  connect(worker, &Work::connectOk, this, [=](){  QMessageBox::information(this,"连接到服务器","连接成功！");  });  connect(worker,&Work::dropConnect,worker,[=](){  t->quit();  t->wait();  worker->deleteLater();  t->deleteLater();  });  t->start();  }  Widget::~Widget()  {  delete ui;  }  void Widget::OnFileOpenError()  {  QMessageBox::warning(this,"错误","选择的文件不能打开！");  }  void Widget::on\_setFile\_2\_clicked()  {  QString ip = ui->ip->text();  unsigned short port = ui->Port->text().toUShort();  emit startConnect(ip, port);  }  void Widget::on\_setFile\_clicked()  {  QString path = QFileDialog::getOpenFileName(this,"选择文件");  if(path.isEmpty())  {  QMessageBox::warning(this,"打开文件","选择的文件路径不能为空");  return;  }  ui->filepath->setText(path);  }  void Widget::on\_send\_clicked()  {  emit sendSignal(ui->filepath->text());  }  void Widget::setValue(int value)  {  ui->progressBar->setValue(value);  } |
| Work.h  #ifndef WORK\_H  #define WORK\_H  #include <QObject>  #include <QTcpSocket>  #include <QFile>  #include <QFileInfo>  #pragma pack(push, 1) //按照1字节对齐  typedef struct packet  {  int len;  char buffer[4096];  }NetWorkPacket;  #pragma pack(pop)  class Work : public QObject  {  Q\_OBJECT  public:  explicit Work(QObject \*parent = nullptr);  //连接服务器  void connectServer(QString ip, unsigned short port);  //发送文件  void send(QString path);  signals:  void connectOk();  void dropConnect();  void canNotOpenFile();  void curPercent(float percent);  private:  QTcpSocket\* m\_tcp;  };  #endif // WORK\_H |
| Work.cpp  #include "work.h"  Work::Work(QObject \*parent)  : QObject{parent}  {  }  void Work::connectServer(QString ip, unsigned short port)  {  m\_tcp = new QTcpSocket;  m\_tcp->connectToHost(QHostAddress(ip), port);  connect(m\_tcp, &QTcpSocket::connected, this, &Work::connectOk);  connect(m\_tcp, &QTcpSocket::disconnected, this,[=](){  m\_tcp->close();  m\_tcp->deleteLater();  emit dropConnect();  });  }  void Work::send(QString path)  {  qInfo() << path;  QFile file(path);  file.open(QIODevice::ReadOnly);  if(file.isOpen())  {  NetWorkPacket p1,p2;  QFileInfo fileInfo(path);  //获取文件大小  int fileSize = fileInfo.size();  //获取文件类型打包  QString fileType = fileInfo.suffix();  QByteArray type = fileType.toLatin1();  int type\_length = fileType.length();;  p1.len = type\_length;  memcpy(p1.buffer,type.data(),p1.len);  p2.len = fileSize;  static int num = 0;  int totalSize = sizeof(int) + p1.len + sizeof(int) + fileSize;  qInfo() << "总大小：" << totalSize;  num += m\_tcp->write((char\*)&p1, sizeof(int) + p1.len);  float percent = (num \* 100) / totalSize;  emit curPercent(percent);  qInfo() << "现在写入：" << num;  while(!file.atEnd())  {  file.read(p2.buffer, sizeof(file.size()));  num += m\_tcp->write((char \*)&p2, sizeof(int) + p2.len);  qInfo() << "现在写入：" << num;  percent = (num \* 100) / totalSize;  emit curPercent(percent);  }  }  else  {  emit canNotOpenFile();  }  } |
| Main  #include "widget.h"  #include <QApplication>  int **main**(int **argc**, char \***argv**[])  {  QApplication **a**(*argc*, *argv*);  Widget **w**;  w.show();  return a.exec();  } |